Estruturas de Repetição

APRESENTAÇÃO

Em muitas situações, não basta que tenhamos somente uma estrutura de seleção em um programa de computador ou a forma sequencial de implementação. **Em resolução de problemas reais, é comum a necessidade de uso de estrutura de repetição,** que seleciona, de certa forma, um trecho de código e faz com que ele seja executado um determinado número de vezes ou quando alguma operação acontecer.

Nesta Unidade de Aprendizagem, você estudará a **construção de um algoritmo** usando praticamente todos os conceitos de programação, com destaque para os **tipos de estrutura de repetição existentes.**

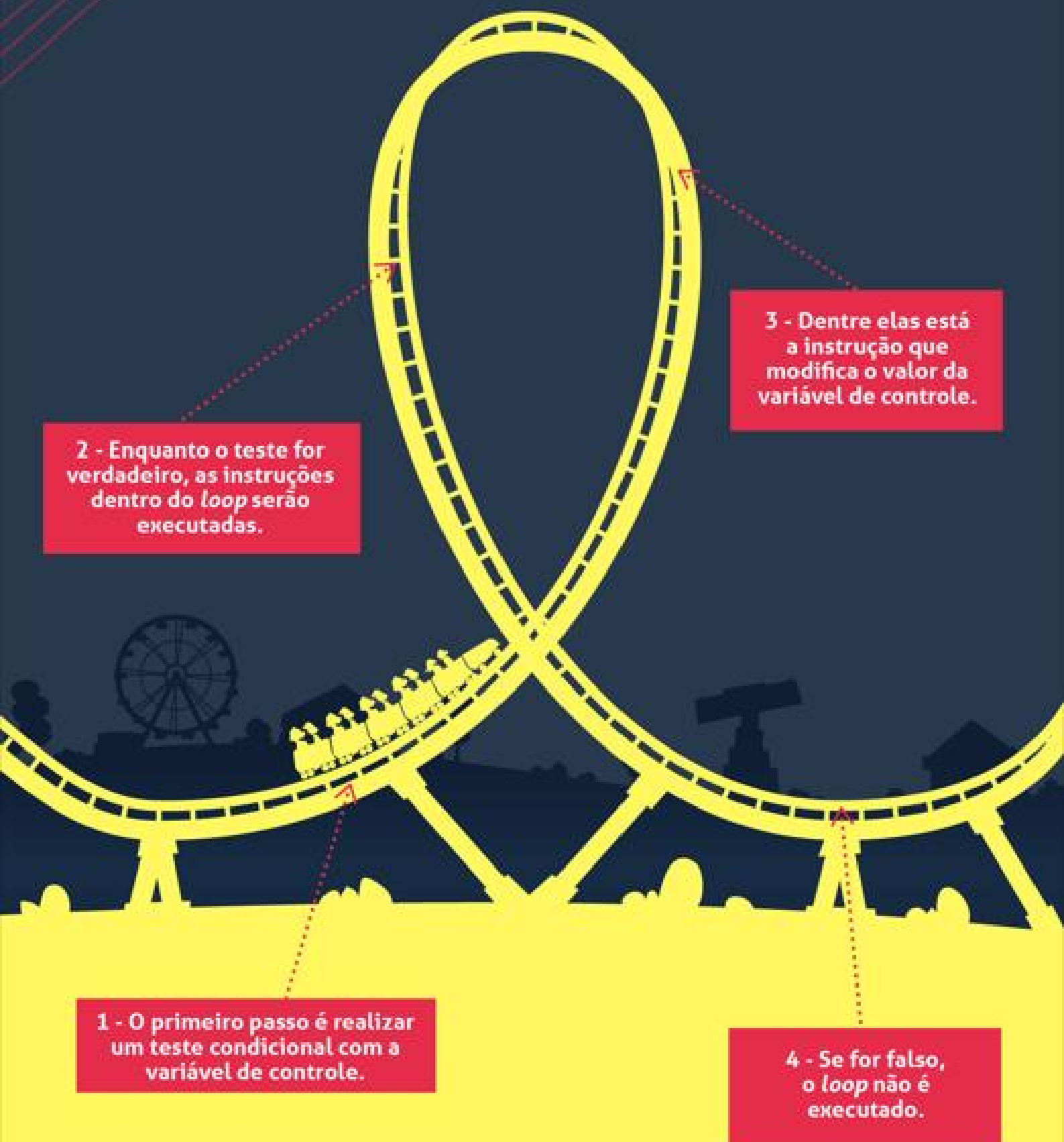
Bons estudos.

Ao final desta Unidade de Aprendizagem, você deve apresentar os seguintes aprendizados:

- Reconhecer o funcionamento da estrutura *while.*
- Implementar programas de computador usando o *for*.
- Entender como a estrutura *do-while* funciona.

INFOGRÁFICO

Quer entender como a estrutura de repetição funciona? No Infográfico, podemos comparar o *loop* da montanha-russa com todos os passos de uma estrutura de repetição.

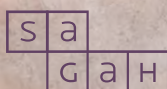
A ESTRUTURA DE REPETIÇÃO *WHILE* NO *LOOP* DA MONTANHA-RUSSA.

CONTEÚDO DO LIVRO

Em programação, muitas vezes é preciso repetir um trecho de código para conseguir resolver um determinado problema ou realizar uma determinada tarefa. Em C, existem três tipos de estrutura de repetição: *while, for e do-while.*

No capítulo Estruturas de repetição, da obra *Algoritmos de programação*, que serve de base teórica para esta Unidade de Aprendizagem, você vai saber mais sobre cada um deles.

Boa leitura.

ALGORÍTMO DE PROGRAMAÇÃO

Marcela Santos

Estruturas de repetição

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Compreender o funcionamento da estrutura *while*.
- Implementar programas de computador usando o *for*.
- Entender como a estrutura *do-while* funciona.

Introdução

Em muitas situações, não basta que tenhamos somente uma estrutura de seleção em um programa de computador, tão pouco somente a forma sequencial de implementação. Em resolução de problemas reais, é comum a necessidade do uso de estrutura de repetição, que seleciona, de certa forma, um trecho de código e faz com que ele seja executo um determinado número de vezes ou quando alguma operação acontecer.

Neste capítulo, você estudará a construção de um algoritmo, usando praticamente todos os conceitos de programação, dando destaque aos tipos de estrutura de repetição existentes.

A estrutura *while*

Existem algumas situações em programação que precisamos repetir o mesmo trecho de código com valores diferentes para as nossas variáveis. Um programa que calcula a média de uma turma de 40 alunos vai repetir a mesma operação 40 vezes: leitura da matrícula do aluno, leituras das notas, cálculo da média e, por fim, mostrar para o usuário o resultado desse processamento de dados.

Da mesma forma que as estruturas de seleção fazem um controle do fluxo de execução, as estruturas de repetição vão mudar o fluxo, repetindo um trecho de código um determinado número de vezes ou quando alguma operação for executada.

Podemos pensar a estrutura de repetição como o *loop* da montanha-russa, onde o carrinho vem em seu fluxo e faz um giro. Em programação, esse *loop* é conhecido como repetição.

O *loop* pode ficar repetindo um número X de vezes. Esse número pode ser determinado pelo programador ou pode ser um valor digitado pelo usuário. Em todos os casos, esse valor ficará armazenado em uma variável que é conhecida como contadora. Um detalhe: você pode dar o nome que quiser para essa variável, mas lembre-se de que é legal o nome da variável ter relação com o significado do que ela armazena.

Além disso, o *loop* também pode ser controlado por algo que o usuário digita ou alguma operação interna a ele. Imagine que o cálculo da média seja digitado até que seja zero para a matrícula do aluno. O *loop,* então, encerraria como resultado de uma operação. Esse tipo de *loop* é obtido por meio de um teste condicional com uma variável, que é conhecida como variável de controle.

Em C, existem os seguintes *loops*: *while*, *for* e *do-while*. Vamos começar com o *while*. Para isso, vamos a um código simples, que mostre os números de 1 a 5, primeiro sem estrutura de repetição. Veja na Figura 1, a seguir.

```
1  #include <stdio.h>
2  int main(){
3      int contador;
4      contador=0;
5      contador=contador+1;
6      printf("%d\n",contador);
7      contador=contador+1;
8      printf("%d\n",contador);
9      contador=contador+1;
10     printf("%d\n",contador);
11     contador=contador+1;
12     printf("%d\n",contador);
13     contador=contador+1;
14     printf("%d\n",contador);
15     return 0;
16 }
```

Figura 1. Código simples sem estrutura de repetição.

Como pode ser visto na Figura 1, repetimos as linha 5 e 6 por 5 vezes, que é a escrita na tela da variável contador. Nesse exemplo, inicializamos a variável contador com 0. Portanto, para que a contagem seja de 1 a 5, é necessário realizar primeiro um incremento de 1 nessa variável para depois exibirmos na tela.

Agora, vamos usar a estrutura de repetição *while* para realizar a mesma tarefa. Veja na Figura 2, a seguir.

```
1  #include <stdio.h>
2  int main(){
3      int contador;
4      contador=0;
5      while(contador<5){
6          contador=contador+1;
7          printf("%d\n",contador);
8      }
9      return 0;
10 }
```

Essa variável será a responsável por controlar a estrutura de repetição

Figura 2. Estrutura de repetição *while*.

As linhas 5 a 8 mostram o uso da estrutura de repetição *while*. Sua sintaxe é a demonstrada na Figura 3:

```
while(condição){
    bloco que ira se repetir;
}
```

Figura 3. Sintaxe da estrutura de repetição *while*.

Alguns detalhes:

- A condição é uma expressão relacional e/ou lógica com a variável de controle.
- A variável de controle deve ser inicializada antes do *loop*.
- A variável de controle precisa ser modificada dentro do *loop* de repetição.

Esses detalhes são importantes, pois, se eles não estiverem presentes no código, o *loop* pode ou não funcionar ou funcionar de forma errada. Para exemplificar, vamos a dois erros bastante comuns que podem acontecer, ou melhor, acontecem com todos que estão aprendendo a programar.

Veja este exemplo, na Figura 4.

```
1  #include <stdio.h>
2  int main(){
3      int contador;
4      contador=6;
5      while(contador<5){
6          contador=contador+1;
7          printf("%d\n",contador);
8      }
9      return 0;
0  }
```

Figura 4. Exemplo de problema de semântica.

Esse programa compila, mas, ao executar, nada acontece — trata-se de um problema de semântica. A linha em destaque é a que inicializa a variável de controle, contador. Ela foi inicializada com o valor 6, e na linha 5 temos a estrutura *while*.

No *while,* é feito um teste condicional, o qual, na primeira rodada, é falso. O fluxo do programa não passa pelo *loop* (delimitado pelo conjunto de chaves) e vai direto para a linha 9, finalizando o código.

Com esse código, deu para perceber que o bloco de operações dentro do *while* só será executado caso, ou melhor, enquanto o teste condicional for verdadeiro. Por isso, cuidado com o valor inicial de sua variável de controle.

Vamos melhorar a sintaxe do *while*, na Figura 5.

```
while(condicao){
    //bloco de operacoes
    //enquanto o teste for verdadeiro
}
```

Figura 5. Sintaxe do *while*.

Esse é o nosso primeiro problema comum em estrutura de repetição. Vamos ao segundo. Veja o código a seguir, na Figura 6.

```
1  #include <stdio.h>
2  int main(){
3      int contador;
4      contador=0;
5      while(contador<5){
6          printf("%d\n",contador);
7      }
8      return 0;
9  }
```

Figura 6. Exemplo de problema de *loop* infinito.

Dessa vez, o código compila, mas tem uma saída diferente: ele fica imprimindo na tela o valor 0, e isso vai ficar acontecendo para sempre ou, melhor dizendo, infinitas vezes. Esse problema é conhecido como *loop* infinito e acontece porque não houve modificação do valor da variável de controle dentro do *loop*, ou seja, o teste condicional vai dar sempre verdadeiro e a linha 6 vai ser executada para sempre.

Então, duas dicas valiosas, ao escrever uma estrutura de repetição no seu programa, são:

- inicializar a variável de controle de forma conveniente.
- ter uma linha dentro do *loop*, que realiza a modificação do valor da variável de controle.

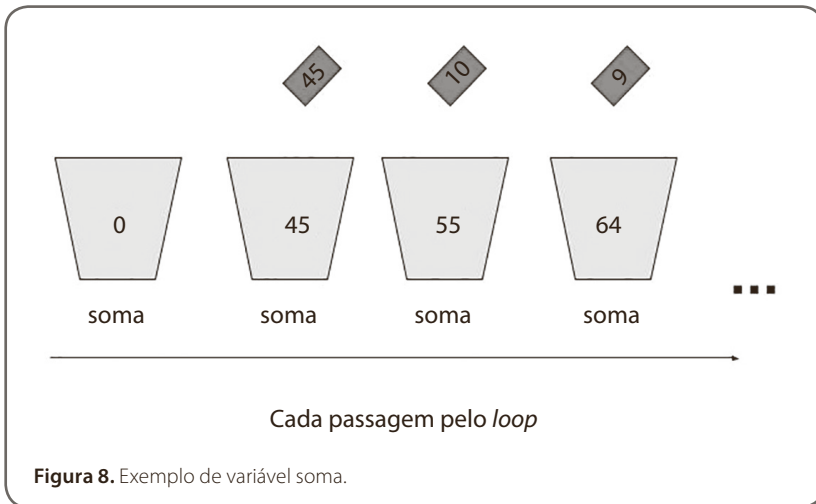
Outro detalhe do uso do *while* é quando não tivermos a quantidade de vezes que a repetição irá acontecer. Veja o seguinte código da Figura 7.

```
1  #include <stdio.h>
2  int main(){
3      int numero,soma;
4      numero=-1
5      soma=0;
6      while(numero!=0){
7          printf("Digite um numero:");
8          scanf("%d",&numero);
9          soma=soma+numero;
10     }
11
12     printf("A soma de todos os numeros digitados eh: %d",soma);
13     return 0;
14 }
15
16 |
```

Figura 7. Exemplo uso do *while* — não há quantidade de vezes da repetição.

Ele soma todos os valores que são digitados. Não temos como saber quantas vezes a pessoa vai digitar um valor válido, mas sabemos que o *loop* irá parar quando o valor digitado, que será armazenado na variável número, for zero, pois o teste condicional é a expressão numero!=0, ou seja, o código irá repetir enquanto o número for diferente de zero.

Outro detalhe para esse código é o uso da variável soma. Esse tipo de uso é o que chamamos de variável, que vai acumulando os valores, bastante útil em programação, como, por exemplo, para calcularmos média de valores. É como se a variável soma fosse uma grande caixa que, na primeira vez que passa pelo código, contém valor 0; depois, todos os números que são colocados lá dentro são somados com o valor que já existia. Veja a Figura 8, a seguir.

Figura 8. Exemplo de variável soma.

Então, o superpoder do *while* é que podemos usá-lo quando não sabemos a quantidade de vezes que o laço vai acontecer — e não se esqueça: o teste condicional do *while* é feito logo do início da sua execução. Vamos agora para o segundo tipo de estrutura de repetição: o *for*.

A estrutura de repetição *for*

A estrutura de repetição *for* também nos auxilia quando precisamos repetir um trecho de código. Sua principal diferença, quando comparada com o *while* e o *do-while* — que veremos a seguir —, é que só pode ser usado quando se sabe a quantidade de vezes que o *loop* vai acontecer.

Vamos avaliar o seguinte código da Figura 9.

```
1  #include <stdio.h>
2  int main(){
3      int numero,soma,count;
4      soma=0;
5      count=5;
6      for (int i = 0; i < count; i++)
7      {
8          printf("Digite um numero:");
9          scanf("%d",&numero);
10         soma=soma+numero;
11     }
12
13     printf("A soma de todos os numeros digitados eh: %d\n",soma);
14     return 0;
15 }
```

Figura 9. Exemplo de código com estrutura de repetição *for*.

Esse código também acumula, somando todos os valores que são digitados pelo usuário. A diferença está no uso da estrutura de repetição *for*. Para que ela seja usada, precisamos definir quantas vezes a repetição irá acontecer — nesse caso, isso é definido com a variável *count*.

A sintaxe da estrutura de repetição *for* é a seguinte:

```
for (int i = 0; i < count; i++)
{
    /* bloco de operaçoes que irao se repetir */
}
```

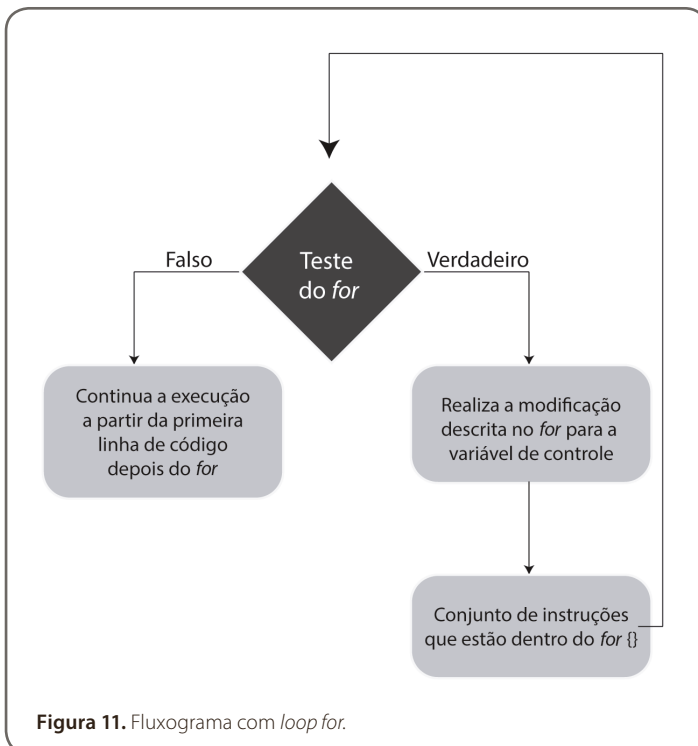
Figura 10. Sintaxe do *for*.

O primeiro valor dentro dos parênteses é a variável de controle — nesse caso, declarada, dentro do próprio *for*, uma prática bastante comum em linguagem C. O segundo termo é o teste condicional, que pode ser traduzido para o seguinte: enquanto o teste for verdadeiro, o *loop* continua; enquanto a variável i for menor que a variável *count* e o último termo, e como a variável i vai ser modificada, ou seja, a instrução que muda o valor da variável de controle, nesse caso, o i será incrementado de um em um.

Os seguintes detalhes devem ser observados:

- A variável de controle não precisa ser declarada no *for*, mas deve levar ao seguinte: quando acabar o *loop*, a variável deixa de existir.
- O teste condicional pode ser feito usando-se qualquer operador lógico e/ou relacional.
- O último termo, também chamado de passo do *loop,* pode ser de incremento, de decremento e do tamanho que for conveniente para o programa.
- O *for* é delimitado por um conjunto de chaves, assim como o *while*.
- O *loop* fica executando enquanto o teste condicional for verdadeiro, ou seja, ao chegar na última linha , o fluxo de execução volta para o teste e verifica se ele é verdadeiro ou falso.

Podemos ver o funcionamento do *loop for* com o seguinte fluxograma, representado na Figura 11:

Figura 11. Fluxograma com *loop for.*

Vamos a mais um exemplo, veja o seguinte código, na Figura 12.

```
1  #include <stdio.h>
2  int main(){
3      int numero,soma,count;
4      soma=0;
5      printf("Quantas vezes vc quer que o loop aconteça?\n");
6      scanf("%d",&count);
7      for (int i = 0; i < count; i++)
8      {
9          printf("Digite um numero:");
10         scanf("%d",&numero);
11         soma=soma+numero;
12     }
13
14     printf("A soma de todos os numeros digitados eh: %d\n",soma);
15     return 0;
16 }
17
```

Figura 12. Código com *loop for.*

Muito parecido com o anterior, exceto pelo valor de *count* ser pedido ao usuário e não atribuído dentro do código. O programa continua sabendo quantas vezes o *loop* vai acontecer, só que agora foi o usuário do programa que definiu. Em ambos os casos, podemos usar o *for,* pois sabemos esse limite. E o *while*? Pode ser usado nesses casos também? A resposta é sim, o *while* é multiuso nesse aspecto. Vamos finalizar nossas estruturas de repetição com o último tipo: o *do-while*.

A estrutura de repetição *do-while*

Para finalizar, vamos à terceira estrutura de repetição: *do-while*. Como vimos anteriormente, existe uma diferença no uso de *while* e *for,* mas ambos têm algo em comum: o teste condicional é feito no início da estrutura. O *do-while* é uma estrutura de repetição onde o teste condicional é feito no fim, ou seja, o bloco de operações será executado pelo menos uma vez, independentemente do valor lógico do teste condicional.

Vamos refazer o exemplo do acumulador de valores, agora usando o *do-while*. Observe a Figura 13.

```
1  #include <stdio.h>
2  int main(){
3      int numero,soma;
4      soma=0;
5      do
6      {
7          printf("Digite 0 para finalizar ou um numero para acumular: ");
8          scanf("%d",&numero);
9          soma=soma+numero;
10     }
11     while(numero!=0);
12
13     printf("A soma de todos os numeros digitados eh: %d\n",soma);
14     return 0;
15 }
```

Figura 13. Exemplo de acumulador de valores com *do-while*.

O *do-while* também funciona quando não sabemos quantas vezes a repetição vai acontecer, assim como o *while*. Um detalhe importante é que o bloco de código que está dentro da estrutura de repetição será executado pelo menos uma vez, visto que o teste condicional é feito somente no fim.

Também é preciso tomar os mesmos cuidados quanto ao *loop* infinito, ou seja, para que o *loop* aconteça de forma conveniente à resolução de um determinando, é preciso que a variável de controle seja modificada dentro do *loop*.

A sintaxe de *do-while* pode ser vista na Figura 14.

```
do{
    //bloco de operaçoes
}
while(teste condicional);
```

Figura 14. Sintaxe de *do-while*.

Para finalizarmos, observe o Quadro 1, com uma comparativa entre as 3 estruturas de repetição existentes na linguagem C.

Quadro 1. Síntese das três estruturas de repetição da linguagem C.

Nome da estrutura	Teste condicional é feito no início ou no fim do *loop*?	Necessidade de se saber quantas vezes o *loop* vai acontecer?
While	Início	Não
For	Início	Sim
Do-while	Fim	Não

Fique atento

Ao se escolher qual estrutura usar, precisa-se analisar o problema e como se deseja resolvê-lo computacionalmente. A prática vai de ajudar sempre nessa escolha. Portanto, programe sempre.

Leituras recomendadas

BOOLEAN type support library. *Cppreference.com*, 2017. Disponível em: <http://en.cppreference.com/w/c/types/boolean>. Acesso em: 22 fev 2018.

MIZRAHI, V. V. *Treinamento em Linguagem C*: curso completo em um volume. 2. ed. São Paulo: Pearson, 2008.

PAES, R. B. *Introdução à Programação com a Linguagem C*. São Paulo: Novatec, 2016.

PINHEIRO, F. A. C. *Elementos de programação em C*. Porto Alegre: Bookman, 2012. 548p.

SCHEINERMAN, E. R. *Matemática Discreta:* uma introdução. São Paulo: Thomson Pioneira, 2003.

DICA DO PROFESSOR

O que é uma estrutura de repetição e como ela funciona? Acompanhe o vídeo e tenha orientações sobre esses conceitos e o modo como a linguagem C trata essas questões.

Conteúdo interativo disponível na plataforma de ensino!

EXERCÍCIOS

1) **O seguinte programa pede para o usuário um número e mostra a tabuada de multiplicação desse número.**

```
1  #include <stdio.h>
2  int main(){
3      int numero,contador,resultado;
4      printf("Qual a tabuada de multiplicacai vc quer saber?");
5      scanf("%d",&numero);
6
7      while(contador<11){
8          resultado=numero*contador;
9          printf("%d x %d: %d\n",numero, contador,resultado);
10
11     }
12 }
```

Foram retiradas as linhas 6 e 10 dessa estrutura, e agora você precisa adicioná-las para um correto funcionamento desse código.

O que deve ser digitado nas linhas 6 e 10?

A) na linha 6: contador=1; na linha 10: contador=contador*1;

B) na linha 6: contador=0; na linha 10: contador=contador+1;

C) na linha 6: contador=1; na linha 10: contador=contador+1;

D) na linha 6: contador=1; na linha 10: contador=contador-1;

E) na linha 6: contador=10; na linha 10: contador=contador+1;

2) **Como podemos reescrever o seguinte trecho de código, utilizando como estrutura de repetição o *for*?**

```
#include <stdio.h>

int main(){

int qtd,contador;

float valor,soma;

soma=0;

contador=1;

printf(" Lista de Compras n");

while(contador<6)

{

printf("Digite a qtd: ");

scanf("%d",&qtd);

printf("Digite o valor por unidade: ");

scanf("%f",&valor);

valor=valor*qtd;

soma=soma+valor;
```

```
contador=contador+1;

}

printf("Valor total da compra: R$ %.2fn",soma);

return 0;

}
```

A)

```
#include <stdio.h>
int main(){
int qtd,contador;
float valor,soma;
soma=0;
printf(" Lista de Compras n");
for (int i = 0; i <5;i++)
{
printf("Digite a qtd: ");
scanf("%d",&qtd);
printf("Digite o valor por unidade: ");
scanf("%f",&valor);
valor=valor*qtd;
soma=soma+valor;
}
printf("Valor total da compra: R$ %.2fn",soma);
return 0;
}
```

B)

```
#include <stdio.h>
int main(){
int qtd,contador;
float valor,soma;
soma=0;
printf(" Lista de Compras n");
for (int i = 0; i <6;i++)
{
printf("Digite a qtd: ");
scanf("%d",&qtd);
printf("Digite o valor por unidade: ");
scanf("%f",&valor);
valor=valor*qtd;
soma=soma+valor;
}
printf("Valor total da compra: R$ %.2fn",soma);
return 0;
}
```

C)

```
#include <stdio.h>
int main(){
int qtd,contador;
float valor,soma;
soma=0;
printf(" Lista de Compras n");
for (int i = 1; i <5;i++)
{
printf("Digite a qtd: ");
scanf("%d",&qtd);
printf("Digite o valor por unidade: ");
scanf("%f",&valor);
valor=valor*qtd;
soma=soma+valor;
}
printf("Valor total da compra: R$ %.2fn",soma);
return 0;
}
```

D)

```
#include <stdio.h>
int main(){
int qtd,contador;
float valor,soma;
soma=0;
printf(" Lista de Compras n");
for (int i = 0; i <5;i++)
{
printf("Digite a qtd: ");
scanf("%d",&qtd);
printf("Digite o valor por unidade: ");
scanf("%f",&valor);
valor=valor*qtd;
soma=soma+valor;
i=i+1;
}
printf("Valor total da compra: R$ %.2fn",soma);
return 0;
}
```

E) Esse tipo de programa não pode ser implementado usando-se o *for*.

3) Em que ocasião ocorre um *loop* infinito?

A) Quando a estrutura de repetição fica executando para sempre.

B) Quando o programa precisa de todo o poder de processamento do computador, algumas vezes chegando até a desligá-lo.

C) Quando a estrutura de repetição não consegue resolver o problema de forma correta, por este ser muito complexo.

D) Quando a variável de controle do *loop* não é inicializada.

E) Quando a repetição nunca é executada .

4) Qual a diferença entre a estrutura *while* e a *do-while*?

A) Na estrutura de repetição *while*, o teste condicional é feito no fim do *loop;* já na estrutura *do-while*, o teste é feito no início.

B) Na estrutura de repetição *while*, não é preciso modificar o valor da variável de controle; já na estrutura *do-while*, essa modificação é obrigatória.

C) Na estrutura de repetição *while*, o teste condicional é feito no início do *loop*; já na estrutura *do-while*, o teste é feito no fim.

D) A *while* só pode ser usada quando se sabe quantas vezes o *loop* será executado; já na *do-while* esse dado não é obrigatório.

E) Não existe diferença alguma entre as duas estruturas.

5) Em que ocasião podemos substituir a *while* pela *for*?

A) Somente quando se sabe quantas vezes a estrutura de repetição será executada.

B) Sempre é possível substituir uma estrutura *while* por uma *for*.

C) Nunca podemos substituir uma estrutura *while* por uma *for*.

D) Somente quando a *while* estiver efetuando operações aritméticas.

E) Somente quando não for preciso inicializar a variável de controle.

NA PRÁTICA

Marcela está, junto com seu time de desenvolvedores, analisando o código de um programa que realiza uma tarefa. **Considera-se a população de duas cidades, A e B, sendo que A tem menos habitantes que B**. O usuário entra com a taxa de crescimento de cada uma delas, e o programa **precisa calcular em quantos anos a população que começou com menos habitantes irá ultrapassar a da outra cidade.**

Durante a análise, ela se deparou com um *loop* infinito. Aponte onde está o erro e como consertá-lo.

Confira o que fez Marcela.

Conteúdo interativo disponível na plataforma de ensino!

O código-fonte precisa ser compilado para poder ser executado: gcc napraticauaer.c

ONDE ESTÁ O ERRO?

```
#include <stdio.h>
int main(){
      float populacaoA = 80000;
      float populacaoB = 200000;
      float taxaCrescA = 0.03;
      float taxaCrescB = 0.015;
      int anos = 1;
      while (populacaoA != populacaoB){
            populacaoA = populacaoA*(1 + taxaCrescA);
            populacaoB = populacaoB*(1 + taxaCrescB);
      anos = anos + 1;
   }
      printf("Populacao A: %.2f\n", populacaoA);
      printf("Populacao B: %.2f\n", populacaoB);
      printf("Anos: %d\n", anos);

      return 0;

}
```

O problema está aqui. Com esse teste, o *loop* vai rodar infinitas vezes.

COMO FOI CORRIGIDO?

```
#include <stdio.h>
int main(){
      float populacaoA = 80000;
      float populacaoB = 200000;
      float taxaCrescA = 0.03;
      float taxaCrescB = 0.015;
      int anos = 1;
      while (populacaoA < populacaoB){
            populacaoA = populacaoA*(1 + taxaCrescA);
            populacaoB = populacaoB*(1 + taxaCrescB);
      anos = anos + 1;
   }
      printf("Populacao A: %.2f\n", populacaoA);
      printf("Populacao B: %.2f\n", populacaoB);
      printf("Anos: %d\n", anos);

      return 0;

}
```

Agora sim: dessa forma, o *loop* vai rodar enquanto a população de A for menor que a de B.

SAIBA MAIS

Para ampliar o seu conhecimento a respeito desse assunto, veja abaixo as sugestões do professor:

Programação em C - PLC0

Assista ao vídeo Me Salva! Programação em C - PLC07 - Comando 'for', para aprender um pouco mais sobre a estrutura de repetição for

Conteúdo interativo disponível na plataforma de ensino!

Unplugged - For Loop Fun - Lesson in Action

Assista ao vídeo Unplugged - For Loop Fun - Lesson in Action, para entender como um loop funciona de forma unplugged, ou seja, sem usar computação.

Conteúdo interativo disponível na plataforma de ensino!

Estruturas de Repetição 1 - Curso de Algoritmos #09

A estrutura de repetição ENQUANTO vai permitir que você execute blocos de comandos várias vezes e simplificar a forma de representar lógicas que vão construir programas.

Conteúdo interativo disponível na plataforma de ensino!

Estruturas de Repetição 2 - Curso de Algoritmos #10

A estrutura Repita..Ate é uma estrutura de repetição com teste lógico no final, o que permite que você execute o bloco interno pelo menos uma vez, independente do resultado do teste.

Conteúdo interativo disponível na plataforma de ensino!

Estruturas de Repetição 3 - Curso de Algoritmos #11

A estrutura Para.. Faça é uma estrutura de repetição com variável de controle, o que permite que você execute o bloco interno uma quantidade determinada de vezes.

Conteúdo interativo disponível na plataforma de ensino!

Hour of Code - Mark Zuckerburg teaches Repeat Loops

Assista ao vídeo Hour of Code - Mark Zuckerburg teaches Repeat Loops, no qual o fundador do Facebook explica como funciona o loop.

Conteúdo interativo disponível na plataforma de ensino!